REMARKS/ARGUMENTS

This Amendment is in response to the Office Action mailed 11/03/2006. Claims 1-36 were pending in this application. Claims 1-36 were rejected. This response amends claims 1, 18, 27-30, and 32-35 and cancels claims 31 and 36. Thus, claims 1-30 and 32-35 are pending.

Rejection under 35 USC § 101

Claims 27-36 were rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. Specifically, it is asserted that the medium is not limited to tangible embodiments, citing paragraph [0042] of the application as filed, which refers to carrier signals. (Office Action dated 11/03/2006, p. 2). Without conceding the merits of the rejection as applied to the original claims, Applicants respectfully submit that the amended claims overcome the rejection.

Although Applicants do not necessarily agree with the rejection, claims 27 and 32 have been amended to recite "a computer program product **stored on** a computer readable medium." To store information, the medium would need to be tangible. These claims, and the claims that depend therefrom, all should constitute statutory subject matter. Applicants therefore respectfully request that the rejection with respect to claims 27-30 and 32-35 be withdrawn.

Claims 31 and 36 were rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. Although Applicants do not necessarily agree with the rejection, claims 31 and 36 have been canceled from the application. Applicants therefore respectfully submit that the rejection with regard to claims 31 and 36 is now moot.

Rejection under 35 USC § 103, *De Armas* in view of *Teilhet*

Claims 1-36 were rejected under 35 U.S.C. §103(a) as being unpatentable over *De Armas* et al. (U.S. Patent No. 6,611,878) (hereinafter "*De Armas*") in view of *Teilhet*. ("Subclassing and Hooking with Visual Basic.") (hereinafter "*Teilhet*"). Claim 1 is allowable as *De Armas* and *Teilhet* either alone or in combination, do not teach or suggest each and every element of claim 1. For example, claim 1 recites:

A method for use by a first process executing in a computer system for interacting with a second process executing in the computer system, the method comprising:

during a startup sequence of the second process, creating a copy of **a global notification hook** of the first process in the second process;

using the copy of the global notification hook, detecting an occurrence of a triggering message passed between an operating system and a thread of the second process;

in response to detecting the occurrence of the triggering message, determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored; and

in the event that the subsequent messages should be monitored, activating a thread-level message hook within the thread of the second process, wherein the thread-level message hook is configured to monitor the subsequent messages. *(emphasis added).*

De Armas cannot render claim 1 obvious, either alone or in combination with *Teilhet*. *De Armas* fails to teach, "in response to detecting the occurrence of the triggering message, determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored," as recited in Claim 1. The rejection asserts that the PostMessage( ) API call is used to send the triggering message. (Office Action, p. 3, item 7).

De Armas is directed to integrating new or modified user interface features or modified functionality to an existing target application program. The PostMessage( ) API call is actually made from the application program, to the operating system 102, which places a message in a thread message queue for the target application. The WH_GETMESSAGE hook is designed for processing or modifying all system messages whenever a GetMessage function is called to retrieve a message from a thread message queue. (*De Armas*, col. 7, lines 33-36). **Upon receipt of such command, the injection DLL 134 initiates the sub-classing process**. (*De Armas*, col. 8, lines 35-36). The Sub-classing process overwrites the original pointer with a replacement pointer to the surrogate window procedure. "By changing the pointer as described above, **any subsequent messages to the target window procedure are dispatched instead to the surrogate window procedure**." (*De Armas*, col. 8, lines 49-53).

Even assuming, for purposes of argument, the PostMessage( ) API call sends the triggering message, and the injection DLL receives the command, thereby detecting the triggering message, *De Armas* fails to teach, "in response to detecting the occurrence of the triggering message, determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored," as recited in Claim 1. Rather,

De Armas teaches that **all subsequent messages are unconditionally dispatched** to the surrogate window procedure. Thus, at most, *De Armas* teaches that all subsequent messages are unconditionally monitored. Since dispatching of subsequent messages is unconditional monitored, *De Armas* does not teach determining **whether** subsequent messages passed between the operating system and the thread of the second process **should be monitored**. Moreover, the triggering message of *De Armas* does not trigger the same act that is recited in Claim 1. The triggering message of Claim 1 is a trigger for the act of determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored. In contrast, the triggering message of *De Armas* is a trigger for the act of initiating the sub-classing process, not for making any determination. Accordingly, the triggering message as taught by *De Armas* does not correspond to the triggering message of Claim 1.

Thus, *De Armas* does not teach or suggest, "in response to detecting the occurrence of the triggering message, determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored," as recited in Claim 1. Further, *De Armas* does not teach, "in the event that the subsequent messages should be monitored, activating a thread-level message hook within the thread of the second process, wherein the thread-level message hook is configured to monitor the subsequent messages," as recited in Claim 1. Thus, *De Armas* cannot render claim 1 obvious.

Moreover, *Teilhet* does not make up for the deficiencies in *De Armas* with respect to claim 1. Although *Teilhet* teaches thread-level hooks and sub-classing a window in a separate process by using the WH_GETMESSAGE hook, *Teilhet* fails to teach, "in response to detecting the occurrence of the triggering message, determining whether subsequent messages passed between the operating system and the thread of the second process should be monitored," as recited in Claim 1. Thus, *Teilhet* cannot render claim 1 obvious, either alone or in combination with *De Armas*. As claim 1 is allowable, dependent claims 2-17 are also patentable for at least the same rationale.

Furthermore, *De Armas* fails to teach, "a global notification hook," as recited in Claim 1. As defined in the specification of the application, "a global notification hook, which

generally includes a hook procedure that receives selected (or all) messages for all application processes but **is not in the message path between the OS and any application process**." (Specification, p. 9, line 1-3). *De Armas* is directed to an approach which allows a technology injection system (TIS) to inject itself directly between a computer operating system and the target program so as to intercept messages and commands to the target program. (*De Armas*, col. 2, lines 49-59). When certain system-wide hooks are installed in the context of a 32-bit Windows operating system, they will cause a designated DLL file to be mapped into the process address space of each application program then running. (*De Armas*, col. 7, lines 8-11). A hook is generally understood as referring to some programming mechanism by which a programmed function can **intercept events, such as messages, before they reach an application**. (*De Armas*, col. 6, lines 64-67). The WH_GETMESSAGE hook is designated for processing or modifying all system messages. (*De Armas*, col. 7, lines 33-36). In Fig. 4 of *De Armas*, the path from the operating system 102 to the target window procedure is interrupted by a direct path from the target message queue 110 to the surrogate window procedure 124.

The hook procedure of *De Armas* intercepts messages between the operating system and the target program. Interception of messages is the primary function of this hook procedure, causing the messages to be dispatched to the surrogate window procedure. Even though the surrogate window procedure is mapped into the process address space of the target application, the target application does not receive the messages intercepted by the hook. As such, the hook procedure of *De Armas* falls directly within the message path between the operating system and the target application program. Accordingly, *De Armas* does not teach or suggest "a global notification hook," as recited in Claim 1. Thus, *De Armas* cannot render claim 1 obvious.

Moreover, *Teilhet* does not make up for the deficiencies in *De Armas* with respect to claim 1. *Teilhet* is directed to describing thread-level hooks which are typically called filter functions. The hook is installed when filter functions are placed into the message stream at a hook point by using the SetWindowsHookEx API function. (*Teilhet*, chapter 3.2, p.1). *Teilhet* also teaches that the system notifies the WH_CBT hook before actions occur. (*Teilhet*, chapter 18.1, p.1). Because the notification occurs before certain actions, the action can be prevented

from occurring by passing back a nonzero return value from the filter function. (*Teilhet*, chapter 18.1, p.2).

Although *Teilhet* teaches the use of the WH_CBT hook both as a thread-level hook and a system-wide hook, the WH_CBT hook of *Teilhet* is **within a message path between the operating system and the application program**. The system notifies the CBT hook of certain actions, such as when a window is created or destroyed, and the CBT hook can prevent the action from occurring. This places the CBT hook, as described in *Teilhet*, within a message path between the operating system and the application program. Accordingly, *Teilhet* fails to teach or disclose a global notification hook, as is recited in Claim 1. Thus, *Teilhet* cannot render claim 1 obvious, either alone or in combination with *De Armas*. As claim 1 is allowable, dependent claims 2-17 are also patentable for at least the same rationale.

Applicants submit that independent claims 18, 27, and 32 also recite features that are not taught or suggested by *De Armas* and *Teilhet* and should be allowable for at least the same rationale as discussed with respect to claim 1. Claims 19-26 depend from independent claim 18 and thus derive patentability at least therefrom. Claims 28-30 depend from claim 27 and thus derive patentability at least therefrom. Claims 33-35 depend from claim 32 and thus derive patentability at least therefrom. Applicants therefore respectfully request that the rejection with respect to the pending claims be withdrawn.

CONCLUSION

In view of the foregoing, Applicants believe all claims now pending in this Application are in condition for allowance. The issuance of a formal Notice of Allowance at an early date is respectfully requested.

If the Examiner believes a telephone conference would expedite prosecution of this application, please telephone the undersigned at 650-326-2400.

Respectfully submitted,

/Naya M. Chatterjee-Marathe/

Naya M. Chatterjee-Marathe
Reg. No. 54,680

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 650-326-2400
Fax: 415-576-0300
Attachments
NMC:mg
60923868 v2